14장. 전역 변수의 문제점

14장. 전역 변수의 문제점

14.1 변수의 생명 주기?

14.1.1 지역 변수의 생명 주기

1. 변수는 생물과 유사하게 생성되고 소멸되는 **생명 주기**(life cycle)가 있다.

2. 전역 변수의 생명 주기 = 애플리케이션의 생명 주기

3. [예제 14-01]

```
function foo() {
    var x = 'local';
    console.log(x); // local
    return x;
}

foo();
console.log(x);   // ReferenceError: x is not defined
```

- 지역 변수 x는 foo 함수가 호출되어 실행되는 동안에만 유효

  → **지역 변수의 생명 주기 = 함수의 생명 주기**

4. 변수의 생명 주기

   : 메모리 공간이 확보된 시점부터, 메모리 공간이 해제되어 가용 메모리 풀에 반환되는 시점

   ➡️ 변수는, 자신이 등록된 스코프가 소멸(메모리 해제)될 때까지 유효.

   누군가 메모리 공간을 참조하고 있으면, 해제되지 않고 확보된 상태?로 남아있게 된다.

5. [예제 14-02]

```
var x = 'global';

function foo() {
    console.log(x); // undefined?
    var x = 'local';
}

foo();
console.log(x);   // global
```

- **호이스팅은 스코프를 단위로 동작**
- **호이스팅**

  : 변수 선언이 스코프의 선두로 끌어 올려진 것처럼 동작하는 자바스크립트 고유 특징

14.1.2 전역 변수의 생명 주기

📄 전역 객체(global object)

: 코드가 실행되기 이전 단계에 자바스크립트 엔진에 의해 어떤 객체보다도 먼저 생성되는 특수 객체

1.

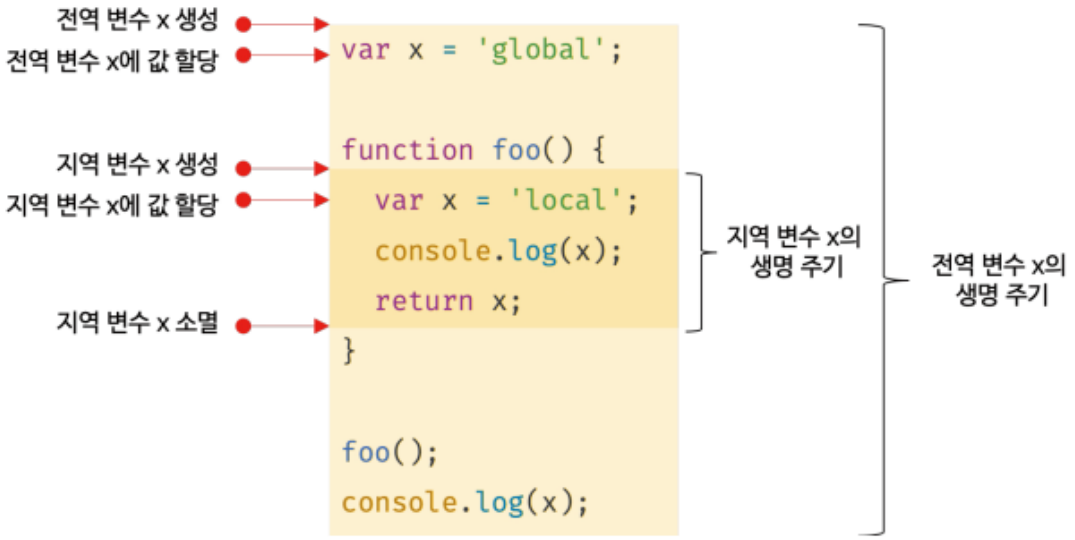

그림 14–2 전역 변수의 생명 주기

출처 : https://velog.io/@gavri/모던-자바스크립트-Deep-Dive-1415-전역-변수의-문제점let-const-키워드와-블록레벨-스코프

- **var 키워드로 선언한 전역 변수의 생명 주기 = 전역 객체의 생명 주기**

14.2 전역 변수의 문제점

1. 암묵적 결합(implicit coupling)

전역 변수를 선언한 의도

→ 전역, 즉 코드 어디서든 참조 & 할당할 수 있는 변수를 사용하겠다는 것.

2. 긴 생명 주기

전역 변수는 생명 주기가 길다.

- 메모리 리소스도 오랜 기간 소비
- 전역 변수 상태 변경할 수 있는 시간도 길고 기회도 多
- var 키워드는 변수의 중복 선언을 허용 → 중복 가능성 有 (의도치 않은 재할당 가능성)

3. 스코프 체인 상에서 종점에 존재

= 변수를 검색할 때 전역 변수가 가장 마지막에 검색된다는 의미

즉, **전역 변수의 검색 속도가 가장 느리다.**

(차이가 그다지 크지는 않지만, 속도의 차이는 분명히 존재)

4. 네임스페이스 오염

자바스크립트의 가장 큰 문제점 中 하나

: 파일이 분리되어 있다 해도 하나의 전역 스코프를 공유한다는 것

→ 다른 파일 내에서 동일한 이름의 전역 변수 or 전역 함수 때문에 **예상치 못한 결과 가능성**

14.3 전역 변수의 사용을 억제하는 방법

전역 변수를 반드시 사용해야 할 이유를 찾지 못한다면 지역 변수를 사용해야 한다.

변수의 스코프는 좁을수록 좋다.

> 전역 변수를 사용하되, 무분별한 전역 변수의 남발은 자제하자.

14.3.1 즉시 실행 함수

1. **모든 코드를 즉시 실행 함수로 감싸면, 모든 변수는 즉시 실행 함수의 지역 변수가 된다.**

14.3.2 네임스페이스 객체

1. 전역에 네임스페이스(namespace) 역할을 담당할 객체를 생성

   → 그다지 유용해 보이지는 X

14.3.3 모듈 패턴

1. 클래스를 모방해서 관련이 있는 변수, 함수를 모아 즉시 실행 함수로 감싸 하나의 모듈로 만듦

2. 캡슐화(encapsulation)

   : 객체의 상태를 나타내는 프로퍼티와, 프로퍼티를 참조하고 조작할 수 있는 동작인 메서드를 하

   나로 묶는 것

3. 정보 은닉(information hiding)

   : 캡슐화는 객체의 프로퍼티나 메서드를 감출 목적으로 사용하기도 함

4. [예제 14-07]

```
var Counter = (function () {
    var num = 0;  // private 변수

    // 외부로 공개할 데이터나 메서드를 프로퍼티로 추가한 객체를 반환
    return {
        increase() {
            return ++num;
        },
        decrease() {
            return --num;
        }
    };
}());

// private 변수는 외부로 노출되지 않는다.
console.log(Counter.num);  // undefined

console.log(Counter.increase()); // 1
console.log(Counter.increase()); // 2
console.log(Counter.decrease()); // 1
console.log(Counter.decrease()); // 0
```

14.3.4 ES6 모듈

1. **ES6 모듈은 파일 자체의 독자적인 모듈 스코프를 제공한다.**

2. [예제 14-08]

```
<script type="module" src="lib.mjs"></script>
<script type="module" src="app.mjs"></script>
```

   모듈 파일의 확장자 - mjs 권장